Computational Techniques and Parallel Software Development for Large-Scale Composite Manufacturing Process Simulations

D. R. Shires, R. V. Mohan,[1] A. Mark, R. Kanapady,[1] and K. K. Tamma[1]
U.S. Army Research Laboratory
Aberdeen Proving Ground, MD 21005

Abstract

Simulation-based design, testing, and acquisition have the potential to positively address and reform some of the ailments found in the Department of Defense acquisition process. Chiefly, these philosophies can be utilized to mitigate the high-cost associated with new weapon systems and speed the product development cycle. In many cases, high-performance and parallel computing resources can assist in these simulation-based endeavors. Composite materials form the basis of many new weapon system developments. Manufacturing process simulations based on physical models and numerical discretizations can be effectively employed to reduce the process and product lead time for the composite material structures, thereby impacting the acquisition process. This paper focuses on computational techniques and parallel software development approaches for large-scale composite manufacturing process simulations for composite structures manufactured by resin transfer molding with particular emphasis on resin impregnation of the fiber preform. Computational techniques play a critical role in the solution of physics-based simulations of large-scale geometrically complex structures within a reasonable computing time. An effective solution technique for the analysis of resin impregnation inside complex mold cavities is described and forms the basis of parallel software developments. Various topics, such as parallel programming paradigms, architectures, languages, parallel software testing, validations, and preliminary performance results of the ongoing parallel software development efforts, are presented. The overall efforts are tailored toward an integrated modeling and testing framework for net-shape composite manufacturing process simulations and product testing in a concurrent engineering environment encompassing nontraditional physics-based high-performance computing.

1 Introduction

The Department of Defense (DOD) current acquisition process often relies upon an antiquated and tedious set of principles, guidelines, and regulations that result in costly material being delivered in an untimely manner. With all its imperfections, however, the system has served the American fighting force rather well. But as new materials form the basis of modern Army weapon systems, new philosophies and ways of thinking are required to reform the acquisition of these weapon systems. Simulation-based design (SBD), simulation-based testing (SBT), and simulation-based acquisition (SBA) form such new philosophies. These

[1]**University of Minnesota, Minneapolis, MN 55455**

philosophies can make effective use of high-performance computing resources, thereby permitting large-scale, real-life modeling and testing in a simulation-based environment. These simulations use models that involve the underlying physical laws, constitutive relations, and robust numerical methodologies designed for optimal performance on high-performance supercomputing architectures. In particular, many new weapon systems employ advanced composite materials made of fiber-reinforced polymer resins. These materials provide a high strength-to-weight ratio, long fatigue life, increased corrosion resistance, and may be incorporated as multi-functional consolidated parts. Defense weapon systems, such as the RAH-66 Comanche, Composite Armored Vehicle (CAV), and the Joint Strike Fighter (JSF), along with dual-use technology in the aerospace and automotive commercial industries, heavily employ these composite material weapon systems. Figure 1 is one illustrative example of the application of composite materials in new military weapon systems.

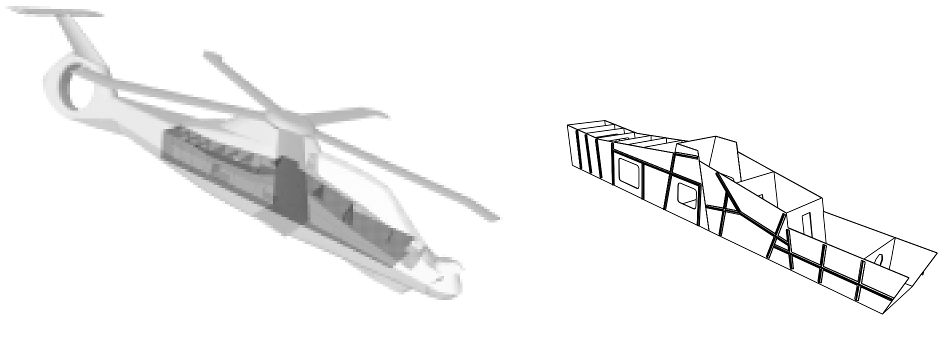

(a) RAH-66 Comanche helicopter. (b) Composite keel beam.

Figure 1: RAH-66 Comanche helicopter and its keel beam.

The prohibitive factor in the development and acquisition of new composite structural weapon systems is not the capability, but rather the cost and limited experience in the manufacturing processes. Of the various structural manufacturing processes, composite net-shape liquid composite molding (LCM) processes such as resin transfer molding (RTM), or its variants such as the Vacuum Assisted RTM, etc., permit large-scale manufacturing of complex composite structures with desired fiber orientations to obtain directional properties in a highly repeatable manner in a production environment. In addition, the process allows building of multifunctional smart structures with embedded sensors for service and damage detection. The process, however, involves repeated trial-and-error testing to determine a working production methodology. This increases the manufacturing process lead time, cost, and acquisition time of the composite material structures and subsequently the weapon systems themselves.

Manufacturing process modeling simulations and analysis, based on representative physical models, provide a simulation-based approach and engineering tools for determining the optimal process parameters and design of tooling and molds. Geometrically and functionally complex composite structural components require accurate modeling of geometric, physical, and material variations that could potentially involve large computational representative physical models composed of several thousands to millions of degrees of freedom. It is impractical and ineffective to conduct such analysis on traditional computational problems. High-performance computing platforms provide the necessary computing power and memory for such large-scale analysis and require the development of optimal computational strategies, implementation, testing, and validation of the appropriate parallel solution software.

This paper focuses on our approaches and experiences in parallel software development using high-performance computing systems. In the implementation and development of parallel software, it is not enough to have optimal parallel software design, programming models, data structures, and interprocessor communication strategies. It is also essential to develop computational algorithms, based on physically accurate representations of mathematical models, that provide the required physical and engineering solutions in optimal time. Here, we are not referring to the choice of one parallel algorithm versus another. We are referring to the development of computational algorithms that will allow for the solution of problems previously unsolvable in a reasonable amount of time even on high-performance computing systems. In this context, a recently developed transient implicit methodology by Mohan et al. [1–4] for the analysis of transient resin impregnation during a pressure-driven flow through porous media has been demonstrated to provide a physically accurate, numerically stable, computationally faster, and algorithmically better solution methodology. This new approach forms the basis of our parallel software developments.

The important consideration from the manufacturing process point of view is the complete impregnation of the fiber preform by the resin. Hence, the first phase of parallel software development focuses on resin impregnation process flow modeling involving transport phenomena of a fluid resin through a porous fiber preform. Once the appropriate parallel software paradigms, optimal data structures, and solution strategies are implemented, the multiphysics phenomena involving thermal and species transport during curing and gelation of polymer systems will be added in future phases.

2 Composite Manufacturing and the RTM Process

Of the various composite manufacturing processes, RTM has a significant potential for cost-effective manufacturing of large net-shape composite structural parts. The process is shown schematically in figure 2. The process employs “fiber preforms,” which are pressed beds of fiber arranged in the shape of the final structure. The preform is placed inside a mold cavity, and a thermosetting polymeric resin is injected under pressure. After the resin impregnates the fiber preform and cures, the finished part is removed from the mold. With proper design, the process can produce high-performance parts with the high fiber volume desired in military and aerospace applications. The industrial experience base in RTM as a complex manufacturing technology is limited. Accordingly, large, complex parts can benefit significantly from large-scale process simulations.

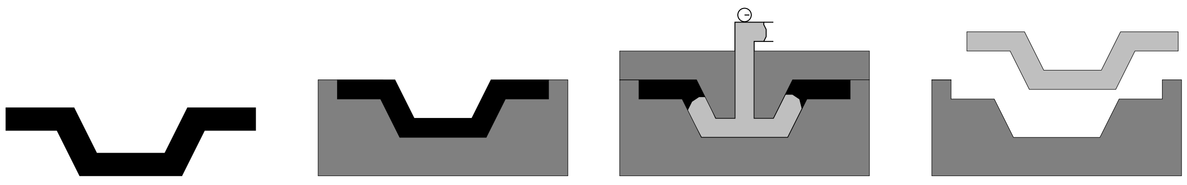

(a) Preform lay-up. (b) Preform placed in mold. (c) Resin injection/curing. (d) Part removal.

Figure 2: Steps in Resin Transfer Molding.

3 Governing Mathematical Model Equations and Computational Strategies for Transport Phenomena in RTM

The governing mathematical equations for various transport phenomena during the RTM process are presented next. As previously stated, the focus in this paper and in the current phase of the parallel software development is on the resin impregnation during the process. Hence, brief discussions of the numerical approaches for the resin impregnation process flow modeling are presented, and the computational advantage of the recently developed pure finite element methodology is illustrated. Discussions of the on-going efforts in the development of computational strategies and serial software for thermal and species transport phenomena with particular emphasis on axisymmetric conditions and experimental validations are discussed elsewhere [5, 6].

3.1 Resin Transport Through Fibrous Porous Media

The macroscopic flow behavior through fibrous porous media is describe by Darcy's law, which relates the average fluid flow rate to the pressure gradient, fluid viscosity, and permeability of the porous medium as

$$\bar{u} = -\frac{\boldsymbol{K}}{\mu} \bigtriangledown P, \tag{1}$$

where $\bar{u}$ is the volume averaged velocity of the resin at a macroscopic scale, μ is the viscosity of the resin, and $\boldsymbol{K}$ is the permeability of the porous fiber preform. Permeability is a measure of easiness of flow through the porous medium and is a function of the fiber architecture and porosity of the medium. It is an important material characteristic involved in the phenomena.

Physically, resin impregnation and flow permeating through a porous media is a transient free surface moving boundary problem, whereby the field equations and the free surface have to be solved and tracked. In mold filling and resin flow impregnation situations such as those seen in RTM, the primary interest is in the temporal progression of resin inside a complex, fixed-mold cavity. The geometric and material complexities of the structural components lead to diverging/merging flow fronts and race tracking effects. Eulerian fixed-mesh approaches are effective in accounting for these complexities in RTM resin impregnation process simulations. The physical distribution of the resin at any instant of time is based on a transient mass balance.

Traditionally, this mass balance has been described by a quasi-steady mass balance defined by

$$\bigtriangledown \cdot \bar{u} = 0. \tag{2}$$

This quasi-steady mass conservation, in conjunction with a finite element-control volume approach using the variable fill factors to compute and track the resin impregnated and dry preform regions, has been traditionally employed in RTM resin impregnation simulations. However, this treatment leads to restrictions on the time step increments between each of the quasi-steady states based on Courant conditions. Such restrictions realistically prevent resin impregnation simulations for large-scale structures by increasing the number of quasi-steady steps during complete impregnation. Accordingly, this solution technique makes such large-scale simulations very much impossible even on high-performance computing platforms.

Another recently developed new approach is based on a transient form of mass conservation equation [1–3] and is called the pure finite element methodology. The finite element developments are based on and account for the transient mass balance of the resin inside a fixed physical and computational domain. The transient mass balance equation at any instant of time inside a general mold cavity is represented as

$$\frac{\partial}{\partial t}\int_{\Omega} \Psi d\Omega = \int_{\Omega} \Psi \bigtriangledown \cdot \bar{u} d\Omega. \tag{3}$$

The resin front and the resin distribution inside the porous fiber region are based on the variable fill factor (Ψ), which denotes the impregnated and dry regions inside the computational domain.

The computational developments by Mohan et al. for the resin flow transport phenomena are based on the introduction of the finite element discretizations for the two unknown variables–namely, the pressure and the fill factor field. The resulting discretized system of equations is solved in an iterative manner until a mass convergence is obtained. The pure finite element methodology is proven to provide a physically accurate, computationally faster, and algorithmically better solution strategy for the finite element modeling of the resin impregnation through the porous media. It has also been well demonstrated for thin, thick, and large-scale composite sections [1–4]. It is to be noted that the computational advantage demonstrated by the pure finite element method when compared with the finite element-control volume technique for RTM resin impregnation simulations is solely due to the computational methodology and the algorithmic solution strategy. The effectiveness of this strategy can be seen in any computational platform, from a desktop personal computer to a high-performance computing system. The pure finite element computational methodology has also made possible large-scale complex resin impregnation simulations [4].

For large-scale process simulations involving high-performance computing systems, the effectiveness of the method is independent of the taxonomy of the parallel processor topologies (interconnection networks, memory hierarchies, etc.) when similar data structures, linear system equation solvers, programming paradigms, and communication strategies are employed in the software development. As an illustration, the normalized computational time (total solution time for the analysis) for complete impregnation of a 10-foot keel beam configuration involving 29,171 nodes and 58,187 elements is shown in Table 1. The computational mesh geometry employed in the simulations and the temporal resin progression contours based on representative injection locations are shown in figure 3. Our experiences with large-scale process simulations involving mesh configurations of higher order indicate that the time step increments between each of the quasi-steady states of the finite element-control volume methodology are extremely small, thereby significantly increasing the computational times and precluding the completion of large-scale structure simulations in a realistic time. With the computational advantage of the pure finite element method well established [1–4], the technique forms the basis of current scalable, parallel software developments.

Table 1: Comparison of computational times: 10-foot keel beam section.

Method	Computational time[a]
Explicit FE-CV	31.28
Pure implicit FE ($\Delta t = 0.5$ sec)	1.00

[a] Relative to the actual computational time corresponding $\Delta t = 0.5$ sec.

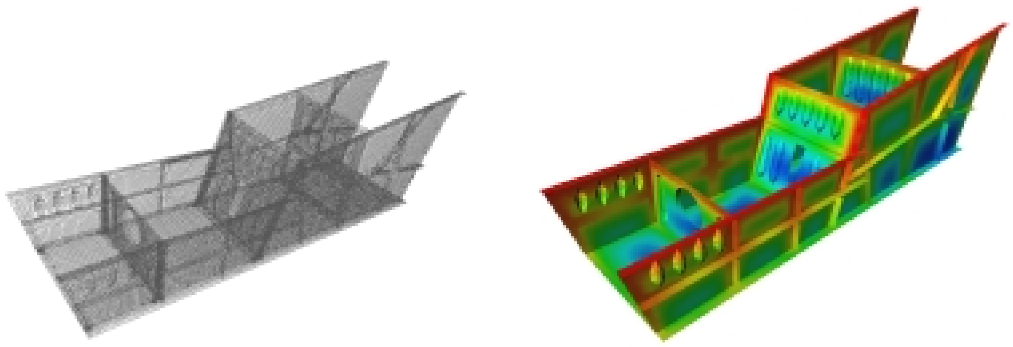


Figure 3: Process simulations: 10-foot keel beam section

4 Parallel Software Development

The governing mathematical equations and the computational approaches of the physical problem have been briefly described in earlier sections. The forthcoming sections focus on the various issues related to parallel software design and development for a scalable, portable, expandable (to add new problem physics), and efficient parallel implementations. Discussions on the taxonomy of parallel processor topologies, programming models, interconnection frameworks, memory hierarchies, and our experiences during the ongoing parallel software development are presented. Demonstrative results from some of the parallel software validations and software acceptance tests (SAT) are also briefly presented.

4.1 On Architectures and Languages

There are several ways to classify parallel programming methodologies. We focus on data parallelism and message passing. Data parallelism exploits the fact that the same operation can be performed concurrently on each item in a set of data. We discuss our experiences in converting a data parallel code written in CM-Fortran to High Performance Fortran (HPF). Message passing most often involves domain decomposition and explicitly defined parallelism realized through calls to a message passing library. We also discuss our current parallel software developments in this context. Each paradigm is applicable to the hardware platforms and topologies widely used today. These architectures include symmetric multiprocessors (SMPs), of which the Sun Ultra E10000 is an example; massively parallel processors (MPPs), of which the CM-5 and Cray T3E are examples; and scalable symmetric multiprocessors (S^{2}MPs), of which the SGI Origin 2000 is an example.

4.2 HPF Data Parallel Programming

The HPF model encompasses both communication and parallelism. It augments Fortran 90, which provides constructs to represent concurrent execution but not domain decomposition. HPF provides additional parallel directives and data placement capabilities. Communication is realized through data distribution, mapping, and alignment. It is the job of the compiler to effectively map and distribute data. Communication is implicit in the code. Parallelism is effected through several mechanisms, including Fortran 90 style array assignments, parallel library routines, the `FORALL` statement, and the `INDEPENDENT` directive [7]. This list is not complete. Extrinsic procedures are available to allow for other programming paradigms or languages. The language continues to evolve through changes to the standard. Unlike MPI, which is realized through calls to a communication library, HPF is a language. To write the most efficient HPF code possible, it is therefore necessary to understand the way an HPF compiler works. The HPF compiler employed during this study is the Portland Group HPF (PGHPF) compiler version 2.4-4 installed at various Major Shared Resource Centers.

4.2.1 Linear System Solver

The composite resin impregnation process simulations involve the solution of a linear system of equations at each of the iterative steps. Traditional finite element computations and sequential software have normally employed an assembly operation of the element-level stiffness matrices into a global system of linear equations that are normally sparse. Direct linear system solvers have been normally employed for resin impregnation process modeling simulations. In a multiprocessor environment, both computational and interprocessor communication aspects have to be taken into account. For any of the programming models and hardware configurations employed, direct solvers normally involve communication during fill-ins. Direct sparse matrix solution techniques involving highly unstructured meshes are not effective for massively parallel computing platforms, except in some special cases involving narrow bandwidth such as those in tridiagonal systems. Hence, iterative solution techniques for the solution of linear system of equations have been employed. The iterative linear system solver involves residuals computed at the element level in the finite element discretization with the convergence criteria determined based on the global level residual of all the degrees of freedom during the solution process. To increase the convergence rate of the linear system of equations, a diagonal preconditioner is employed. Detailed discussion of the iterative system of equations based on a matrix-free element-level iterative solver can be found elsewhere [4].

4.2.2 Experiences in Code Conversion

The HPF compiler attempts to support data parallelism found in prior data parallel languages. It supports and translates function calls from CM-Fortran. However, we found that several changes were required and warranted with our pre-existing CM-Fortran data parallel code. Two areas are of special importance. The first deals with array syntax. In the past, many data parallel languages relied heavily on the use of array syntax to describe parallelism. However, a major drawback to this syntax is that many temporary multidimensional arrays are often required. Memory requirements can quickly get out of hand for even small problems. By using HPF directives and coding structure, several large multidimensional arrays were replaced with scalars. Scalar replacement can have an enormous beneficial impact on cache-based architectures.

The second and most important change deals with gather and scatter calls. The CM-Fortran code used the Connection Machine Scientific Software Library (CMSSL), which was created for array syntax notation and data parallel architectures. The CM-Fortran code used the CMSSL routines `part_scatter_setup`, `part_scatter`, `part_gather_setup`, and `part_gather`. These routines were used to perform partitioned scatter and gather operations, respectively. These routines are necessary for communicating nonlocal data. In the case of scattering, these routines use a source array, a destination array, and a pointer array containing the scattering pattern. Data are scattered from the source array to the destination array.

The CMSSL routine `part_gather_setup` was available to optimize data locality and reduce the associated communication time. During the setup phase, this routine analyzed the gathering pattern that was supplied and reordered the pointer array to achieve better data locality. In HPF, there are no default library routines to do gather operations. The CMSSL equivalent of the two calls:

```
call part_gather_setup(lm,.true.,fillfac,setup,ier)
call part_gather(elfill,fillfac,.true.,setup)
```

can be performed by nested `INDEPENDENT` do-loops:

```
!hpf$ independent, new(j)
       do i = 1, ndel
!hpf$ independent
          do j = 1, nelem
             elfill(i,j) = fillfac(lm(i,j))
          enddo
       enddo
```

The scatter operation is slightly different. Here are the example CMSSL routines to perform a scatter operation:

```
call part_scatter_setup(lm, .true., wgnode, setup, ier)
call part_scatter(wgnode, wel, .true., setup)
```

There is an HPF library function `sum_scatter` to perform the reduction:

```
wgnode = sum_scatter(wel, wgnode, lm)
```

The full details of the implementation are hidden, but most likely this call computes a communication schedule for data going to and arriving from remote nodes, moves the data, and then computes the reduction. It is also possible for reductions to be performed locally before sending out the data.

Communication operations can be very expensive. A profile of our code revealed that it was communication-bound with well over 50% of the execution time spent in calls to the `sum_scatter` library routine. Approximately 20% of the time was spent in code segments performing gather operations. The library routine `sum_scatter` is called repeatedly, thousands of times even for small problems. Since it is a library routine, our assumption was that each time it was called, a schedule was being computed, executed, and any information gathered by the scheduling algorithm was being discarded before the next call. This was confirmed through personal communication with the PGHPF compiler group [8].

The ability to reuse communication schedules is essential to getting good performance with this code. The Portland Group has already recognized this need. They provided us with an experimental release 2.4-dev99a of their HPF compiler. The Cray T3E is the only computer currently targeted in this release. This version of the software allows the programmer to store and reuse a pointer to the communication schedule determined by the compiler. The schedule can be called repeatedly, thereby removing the need to recompute the schedule at each call to `sum_scatter`. While the details of the communication computation are hidden, it is easy to envision a nonoptimal scheduling algorithm taking at least $O(n^2)$ time, with n being the number of elements in a finite element mesh. The call to `alpkm1 = sum_scatter(aelpk, alpkm1, lm)` is replaced with:

```
       sked = pghpf_comm_sum_scatter_2(tfill,yl,.true.,lm,.true.)
       ...
       call pghpf_comm_execute(sked, alpkm1, aelpk)
       ...
       call pghpf_comm_free(1,sked)
```

The Portland Group reports that some users have experienced a three-fold code speedup after switching to reusable schedules [8]. In our case, the time to compute the scatter reduction was approximately cut in half. Currently, only the schedules for scatters may be reused. While gather operations do not contain the arithmetic reduction, they are equally problematic. Gather operations generate a lot of code to compute off-node data locations. In test runs, the gathers started to take longer than the scatter operations.

The ability to reuse communication schedules remains a vendor-specific feature as the HPF 3.0 standard does not address the issue. It will undoubtedly be incorporated into future standards. The ability to reuse communication schedules inside of independent do-loops is also under investigation. Syntax to accomplish this has been proposed by the Japan Association for HPF(JAHPF) [8].

4.3 Message Passing using MPI

An effective parallel programming model providing good portability across different high-performance computing architectures is based on MPI communication calls. The computational problem and the nature of the numerical discretization employed permit the decomposition of the problem domain into several subdomains, with each assigned to a different computational processing element. Domain independent computations are carried out in parallel in each of the subdomains without need for any communications between the processors. Communications are involved when interface conditions and global quantities are involved and are performed using MPI communication calls. Parallel software developments based on this approach are currently in progress.

4.3.1 Parallelism in MPI

Unlike HPF, which attacks parallel programming from a higher conceptual level, MPI is known as explicit parallel programming since the programmer must instrument the code with appropriate communication calls to achieve the desired parallelism. This has both good and bad connotations. Well-written code using few synchronous communications and blocking calls can be very efficient. Conversely, the compiler cannot optimize poorly structured communication since this is done strictly through user calls to library routines. MPI most closely associates with the Single Program-Multiple Data (SPMD) model of parallel computing. Each processor has a copy of the program with local storage space for variables. There are no shared variables amongst the processes. Each process works with unique data, and all variables and data are communicated by explicit calls to a runtime messaging library. MPI is probably the most widely used parallel programming method today and, as such, is very portable across platforms [9].

Data placement, the partitioning of data to allow for parallel execution, is performed through directives in HPF codes. MPI provides no such functionality. Instead, the user must partition data sets in a way to limit communication requirements between the various subdomains. This partitioning problem is in no way trivial. Fortunately, various utilities have been developed to perform this function. We are currently using METIS and ParMETIS for domain decomposition [10]. Figure 4 illustrates the different partitioned subdomains of a complex 24-foot keel beam of the RAH-66 Comanche based on a 16-processor partition. The first step in the parallel software development using the MPI-based approach is setting up appropriate data structures and formulating data sets for the different subdomains and processors based on the global mesh configuration. It is also necessary to know the neighboring processors and the interface nodes that are shared. This permits setting up appropriate communication patterns, synchronization, and load balancing across the processors. Interprocessor communication is needed whenever global quantities are required. The linear system solvers again employ preconditioned iterative solvers based on matrix-free element-level conjugate gradient residuals.

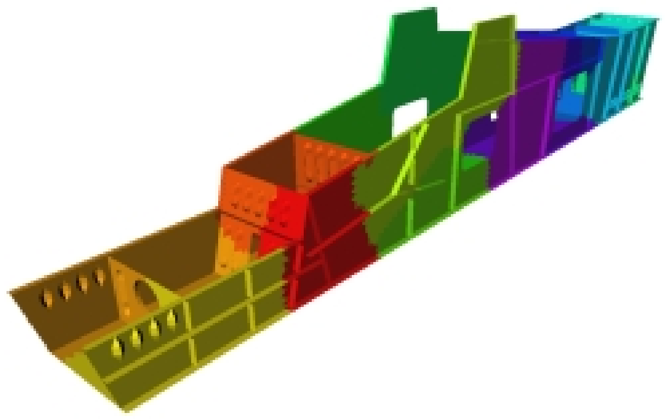


Figure 4: A 16-processor partition of the 24-foot keel beam.

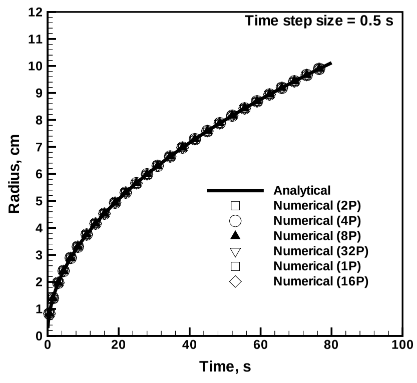
MPI utilizes native compilers and is far less proprietary than many HPF compilers. While complicating the software development cycle, it does open up the possibility for more enhanced profiling and optimization. Because of the SPMD nature of MPI codes, fine tuning for parallel execution falls naturally from serial optimization since multiple copies of the code will be executing. A complete discussion of the parallel software development, validation, testing, and performance of the MPI-based domain decomposition approach as compared to the HPF based approach will be disseminated in future publications.

5 CHSSI-SAT Requirements for Parallel Software Development

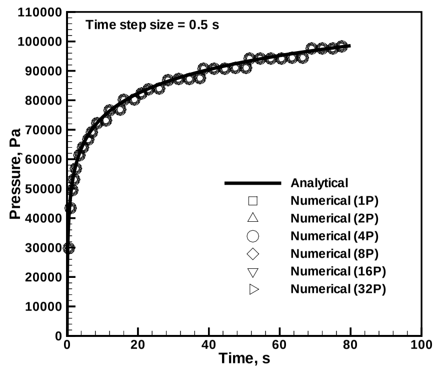
The Common High-Performance Scalable Software Initiative (CHSSI) program, under which the current parallel software development efforts are performed, requires the testing and validation of the parallel software code with the results from analytical solutions, numerical results from sequential codes, and experimental data. We briefly present some of the results from testing and validation of the parallel software developments for illustrative purposes.

5.1 Radial Injection - Constant Flow Rate Injection

A simple circular plate configuration with radius R and a hole of radius R_0 at the center, forming the injection gate of the mold, is considered. The inner radius of the plate is subjected to a constant flow rate injection condition. Being a simple geometry, analytical solutions are available for the temporal location of the progressing front and the inlet injection pressure [1,2]. Comparisons of temporal resin front locations and inlet injection pressures obtained analytically and from the parallel software development code (employing multiple processor configurations) are shown in Figure 5. The comparisons clearly indicate that the numerical results from the multiprocessor parallel software matches the analytical and single processor results exactly. In fact, the physical and numerical trends seen in a single processor solution are also seen in multiprocessor parallel software. Other comparisons to validate and test the parallel software with experimental data and single processor composite structural solutions have been performed. However, they are not presented here due to space considerations. The numerical results obtained from the parallel software code are in complete agreement. Efforts to optimize and fine-tune the code are in progress. Timing and scalable studies continue, and results will be disseminated in the future.



(a) Flow front comparisons



(b) Injection pressure comparisons

Figure 5: SAT validation: comparisons with analytical solution.

5.1.1 Preliminary Performance Data (based on PGHPF version)

Since the ability to reuse communication schedules is only supported on the PGHPF compiler for the Cray T3E, our discussions are currently limited to that architecture. The wall-clock execution time for a particular problem and related processor counts is given in Table 2. The execution times are based on a multiuser mode with other users and processes running concurrently. Time constraints and machine access limited the number of currently reported preliminary results. We profiled the code and concentrated on optimizing the most time consuming part of the algorithm. However, these results are highly preliminary. We are in the process of better analyzing these numbers and collecting more runtime data to determine more appropriate compiler options, data layouts, and memory management within the code. As more processors are employed, the reduction in execution time indicates that the code is highly parallel in nature. Further investigations regarding memory usage inside PGHPF, etc., are required to optimize and fine-tune the parallel software. Amdahl's law appears to have negligible impact on explaining the decreasing efficiency as more processors are added. The limits to speedup appear to reside wholly in areas of the code requiring interprocessor communication. As such, the ability to perform precommunication reductions and reuse communication schedules appears to be the best opportunity at increasing code speed. *It is to be noted that the timings reported here are highly preliminary. Further profiling, optimization, testing, and a potential expanded reuse of communication schedules in the parallel software should provide even greater speedup. This work is currenty in progress.*

Table 2: Wall-clock execution times (in seconds) of data parallel Pure Finite Element code on the Cray T3E-1200.

Problem	Number of processors		
	8	16	32
RAH 10-foot keel beam[a]	8515.41	6111.37	3603.69

[a] 29171 nodes, 58187 elements, time step size = 0.5 sec.

6 Concluding Remarks

High-performance and parallel computing resources can assist in the simulation-based acquisition endeavors of new weapon systems. These require the development of appropriate parallel high-performance computing software for concurrent engineering analysis of process, product design, and testing of new materiel systems. In particular, with composite materials forming the basis of many new weapon systems, the current efforts in the development of computational techniques and appropriate parallel software for large-scale composite manufacturing process simulations were discussed. Physical mathematical models based on underlying physical phenomena form the foundation of the process simulations. The current phase of parallel software development focuses on the resin impregnation transport in dry fiber porous media during manufacture of net-shape composite structures by RTM. Computational techniques play a critical role in the solution of physics-based simulations of large-scale, geometrically complex structures in a realistic computing time. A novel transient finite element technique to simulate the resin impregnation behavior during composite manufacturing by the RTM process was briefly highlighted. The pure finite element computational algorithm permits large-scale simulations to be completed in a realistic time and assist in the production of large-scale, geometrically complex composite parts. The current efforts and experiences in the development of parallel software for large-scale resin impregnation process simulations were discussed and presented. Work is in progress toward code profiling and fine tuning for optimization of PGHPF parallel code and in the development of appropriate explicit message passing parallel software. Future efforts will involve a comparison of these two approaches for parallel software development. The overall efforts are geared toward providing an integrated modeling and testing framework for net-shape large-scale composite structures.

References

[1] R. V. Mohan, N. D. Ngo, and K. K. Tamma, "On a pure finite-element-based methodology for resin transfer mold filling simulations," *Polymer Engineering and Science*, vol. 39, January 1999.

[2] R. V. Mohan, N. D. Ngo, K. K. Tamma, and K. D. Fickie, "On a pure finite element based methodology for resin transfer mold filling simulations," in *Numerical Methods for Thermal Problems* (R. W. Lewis and P. Durbetaki, eds.), vol. IX, (Atlanta, GA), pp. 1287–1310, Pineridge Press, July 1995.

[3] R. V. Mohan, N. D. Ngo, K. K. Tamma, D. R. Shires, and K. D. Fickie, "Process modeling and implicit tracking of moving fronts for three-dimensional thick composites manufacturing," in *AIAA-96-0725, 34 th Aerospace Sciences Meeting*, (Reno, NV), January 1996.

[4] R. V. Mohan, D. R. Shires, A. Mark, and K. K. Tamma, "Advanced Manufacturing of Large Scale Composite Structures: Process Modeling, Manufacturing Simulations and Massively Parallel Computing Platforms," *Journal of Advances in Engineering Software*, vol. 29, no. 3-6, pp. 249–264, 1998.

[5] N. D. Ngo, R. V. Mohan, P. W. Chung, K. K. Tamma, and D. R. Shires, "Developments encompassing non-isothermal/isothermal liquid composite molding process modeling/analysis: Computationally effective and affordable simulations and validations," *Journal of Thermoplastics - Special Issue on Affordable Composites*, vol. 11, November 1998.

[6] N. D. Ngo, R. V. Mohan, K. K. Tamma, and D. R. Shires, "Finite element techniques for the analysis of transport phenomena during composites manufacturing," in *21st Army Science Conference - Science & Technology for Army After Next*, (Norfolk, VA), pp. 711–716, June 1998.

[7] C. H. Koelbel, D. B. Loveman, R. S. Schreiber, G. L. Steele Jr., and M. E. Zosel, *The High Performance Fortran Handbook*. The MIT Press, 1994.

[8] Portland Group, 1999. Private Communication.

[9] P. Pacheco, *Parallel Programming with MPI*. Morgan Kaufmann, 1997.

[10] G. Karypis and V. Kumar, *METIS: A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices*. University of Minnesota and the Army HPC Research Center, 1997.